

How To Increase Test Coverage (and Confidence!) with Mayhem

Hosted by Lakshmia Ferba & Vincent Lussenburg



Overview

1. House Rules
2. Testing pyramid
3. What is test coverage?
4. System (API) under test
5. Test types: unit, integration, end to end
6. Demo of reliability testing + coverage measurement
7. Conclusion

House Rules

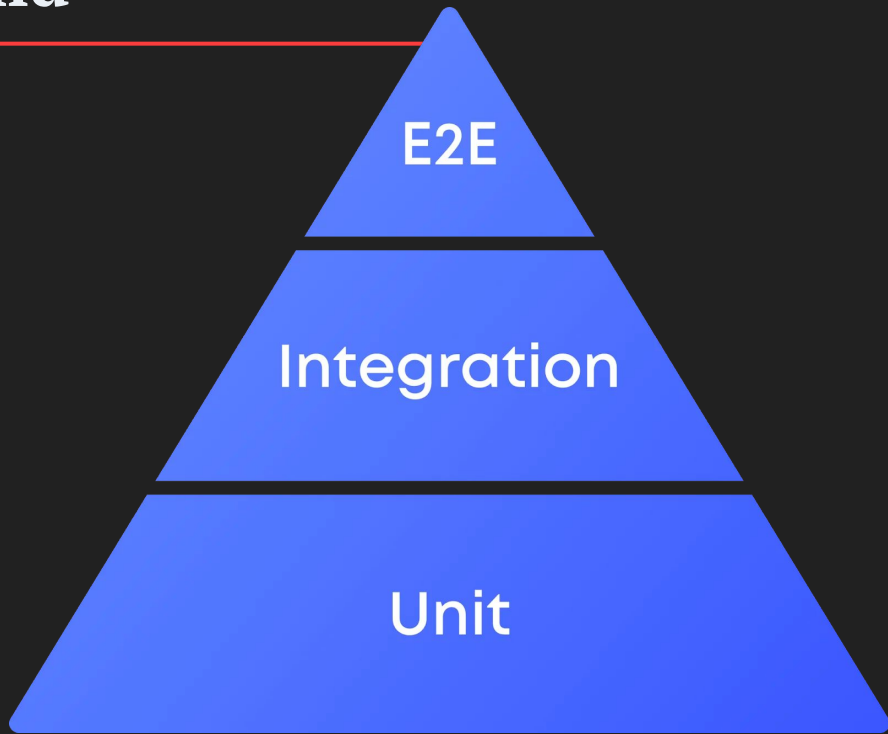
- Please be respectful and kind to the hosts and participants of this webinar.
- Feel free to send us questions as they come up.
- In order to qualify for the raffle, you must attend the entire webinar and not fall under one of these categories:
 - Government Employee
 - Government Contractor

Software Testing: The Pyramid

- Trade offs going up:
 - **MORE** effort to write/maintain test
 - **LONGER** feedback loop
 - **MORE** brittleness/flakiness
 - **HIGHER** level of “realism”

Testing tips (Mike Cohn, Kent Beck):

- Write tests with different granularity
- The more high-level you get the fewer tests you should have
- Timebox important tests in the apex
- Don't unit test trivial code (Kent Beck)
 - (but DO cover them somewhere else -VL)



What is test coverage?

This is what ChatGPT told me:

- Test coverage refers to the extent to which a software application is tested by a set of test cases.
- It measures the effectiveness of testing by identifying the percentage of code or functionality that is exercised during testing.
- Test coverage helps ensure that all parts of an application are tested, reducing the risk of defects and improving software quality.

System (API) Under Test



Mattermost is an open source platform for secure collaboration across the entire software development lifecycle.

<https://github.com/mattermost/mattermost-server>

(AKA, a Slack clone)

Unit Test Coverage

Important code is covered, some permission exceptions verified

```
func TestRestoreChannel(t *testing.T) {
    th := Setup(t).InitBasic()
    defer th.TearDown()

    publicChannel1 := th.CreatePublicChannel()
    th.Client.DeleteChannel(publicChannel1.Id)

    privateChannel1 := th.CreatePrivateChannel()
    th.Client.DeleteChannel(privateChannel1.Id)

    _, resp, err := th.Client.RestoreChannel(publicChannel1.Id)
    require.Error(t, err)
    CheckForbiddenStatus(t, resp)

    _, resp, err = th.Client.RestoreChannel(privateChannel1.Id)
    require.Error(t, err)
    CheckForbiddenStatus(t, resp)

    th.TestForSystemAdminAndLocal(t, func(t *testing.T, client *model.Client) {
        defer func() {
            client.DeleteChannel(publicChannel1.Id)
            client.DeleteChannel(privateChannel1.Id)
        }()
    })
}
```

```
github.com/mattermost/mattermost-server/v6/server/channels/api4/channel.go (79.6%)
not tracked  no coverage  low coverage  * * * * * * * * *  high coverage
func restoreChannel(c *Context, w http.ResponseWriter, r *http.Request) {
    c.RequireChannelId()
    if c.Err != nil {
        return
    }

    channel, err := c.App.GetChannel(c.AppContext, c.Params.ChannelId)
    if err != nil {
        c.Err = err
        return
    }
    teamId := channel.TeamId

    auditRec := c.MakeAuditRecord("restoreChannel", audit.Fail)
    defer c.LogAuditRec(auditRec)
    auditRec.AddEventPriorState(channel)

    if !c.App.SessionHasPermissionToTeam(*c.AppContext.Session(), teamId, model.PermissionManageTeam) {
        c.SetPermissionError(model.PermissionManageTeam)
        return
    }

    channel, err = c.App.RestoreChannel(c.AppContext, channel, c.AppContext.Session())
    if err != nil {
        c.Err = err
        return
    }

    auditRec.AddEventResultState(channel)
    auditRec.AddEventObjectType("channel")
    auditRec.Success()
    c.LogAudit("name=" + channel.Name)

    if err := json.NewEncoder(w).Encode(channel); err != nil {
        c.Logger.Warn("Error while writing response", mlog.Err(err))
    }
}
```

End to End Test Coverage

Positive test cases (mostly)

More expensive maintained (cypress)

Tests webapp + api (takes long to run)

```
it('should go to self direct channel using the multiple ways to go', () => {
  // # Create a self direct channel
  cy.apiCreateDirectChannel([testUser.id, testUser.id]).then(({channel: ownDMChannel}) => {
    // # Visit the channel using the channel name
    cy.visit(`/ ${testTeam.name}/channels/${testUser.id}__${testUser.id}`);

    // * Check you can go to the channel without problem
    cy.get('#channelHeaderTitle').should('be.visible').should('contain', `${testUser.username}`);

    // # Visit the channel using the channel id
    cy.visit(`/ ${testTeam.name}/channels/${ownDMChannel.id}`);

    // * Check you can go to the channel without problem
    cy.get('#channelHeaderTitle').should('be.visible').should('contain', `${testUser.username}`);

    // # Visit the channel using the username
    cy.visit(`/ ${testTeam.name}/messages/@${testUser.username}`);

    // * Check you can go to the channel without problem
    cy.get('#channelHeaderTitle').should('be.visible').should('contain', `${testUser.username}`);

    // # Visit the channel using the user email
    cy.visit(`/ ${testTeam.name}/messages/${testUser.email}`);

    // * Check you can go to the channel without problem
    cy.get('#channelHeaderTitle').should('be.visible').should('contain', `${testUser.username} (you)`);
  });
});
```

```
github.com/mattermost/mattermost-server/v6/server/channels/api4/channel.go (35.6%)
}
not tracked not covered covered
}

func restoreChannel(c *Context, w http.ResponseWriter, r *http.Request) {
  c.RequireChannelId()
  if c.Err != nil {
    return
  }

  channel, err := c.App.GetChannel(c.AppContext, c.Params.ChannelId)
  if err != nil {
    c.Err = err
    return
  }
  teamId := channel.TeamId

  auditRec := c.MakeAuditRecord("restoreChannel", audit.Fail)
  defer c.LogAuditRec(auditRec)
  auditRec.AddEventPriorState(channel)

  if !c.App.SessionHasPermissionToTeam(*c.AppContext.Session(), teamId, model.PermissionManageTeam) {
    c.SetPermissionError(model.PermissionManageTeam)
    return
  }

  channel, err = c.App.RestoreChannel(c.AppContext, channel, c.AppContext.Session())
  if err != nil {
    c.Err = err
    return
  }

  auditRec.AddEventResultState(channel)
  auditRec.AddEventObjectType("channel")
  auditRec.Success()
  c.LogAudit("name=" + channel.Name)

  if err := json.NewEncoder(w).Encode(channel); err != nil {
    c.Logger.Warn("Error while writing response", mlog.Err(err))
  }
}
```




Mayhem: ML-Driven Integration Testing



Self-learning test creation

Every result generates new conditions, inputs, and tests



Fuzzing

Coverage-guided heuristics that use instrumentation to generate new tests.

Does “x” execute any new code?

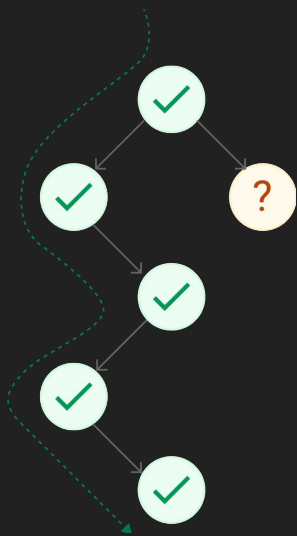
Does “y” execute any new code?

...

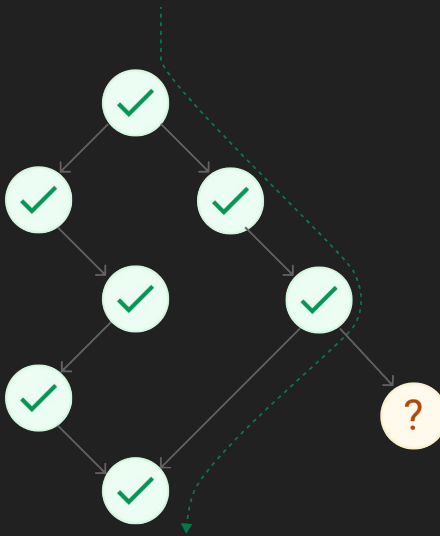
Automated triage and regression test creation

Results are deduplicated and regressions automatically created

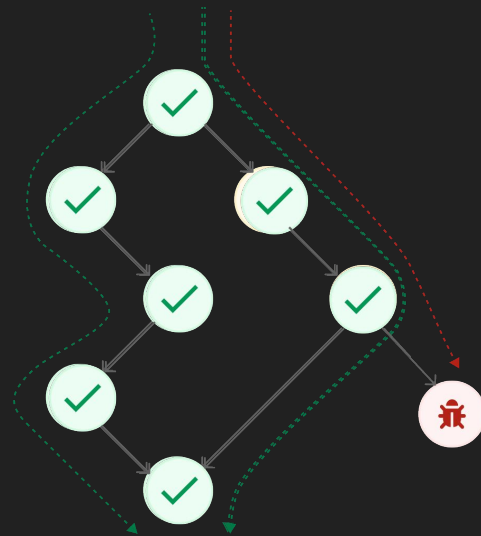
When code explodes, diagnose vulnerability.
Iterate until everything is tested.



(1)



(2)



(3)

Demo

```
# New Go1.20 functionality to measure coverage on running applications!
GOFLAGS=-cover GOCOVERDIR=covdatafiles make test-data

# run Mayhem against the locally started server
mapi run forallsecure/mattermost-webinar 1m openapi.json \
    --url "http://localhost:8065/api/v4" \
    --include-endpoint "^(GET|POST) /channels$" \
    --include-endpoint "^(GET|PUT|DELETE) /channels/{channel_id}$" \
    --include-endpoint "^POST /channels/{channel_id}/restore$" \
    --header-auth "Authorization: Bearer ihutw4wq7prg5mwfu5fubnwqda" \
    --resource-hint "channel_id:ryf6em74pjfn8f3mtsxax3idhw" \
    --interactive

# Stop the server, signaling to write all coverage data to files
make stop

# Convert to cobertura format (great for upload to codecov.io)
go tool covdata textfmt -i=covdatafiles -o=coverage-fuzz.out


# Convert to HTML
go tool cover -html=cov-fuzz.out
```

Mayhem Code Coverage!

See something that was not covered in unit tests is covered here, but also the other way around!

Run time is generally less than e2e-tests (granted, e2e-tests also test the UI)

No code or test suites to maintain!



```
github.com/mattermost/mattermost-server/v6/server/channels/api4/channel.go (19.7%)
not tracked  not covered  covered
func restoreChannel(c *Context, w http.ResponseWriter, r *http.Request) {
    c.RequireChannelId()
    if c.Err != nil {
        return
    }

    channel, err := c.App.GetChannel(c.AppContext, c.Params.ChannelId)
    if err != nil {
        c.Err = err
        return
    }
    teamId := channel.TeamId

    auditRec := c.MakeAuditRecord("restoreChannel", audit.Fail)
    defer c.LogAuditRec(auditRec)
    auditRec.AddEventPriorState(channel)

    if !c.App.SessionHasPermissionToTeam(*c.AppContext.Session(), teamId, model.PermissionManageTeam) {
        c.SetPermissionError(model.PermissionManageTeam)
        return
    }

    channel, err = c.App.RestoreChannel(c.AppContext, channel, c.AppContext.Session)
    if err != nil {
        c.Err = err
        return
    }

    auditRec.AddEventResultState(channel)
    auditRec.AddEventObjectType("channel")
    auditRec.Success()
    c.LogAudit("name=" + channel.Name)

    if err := json.NewEncoder(w).Encode(channel); err != nil {
        c.Logger.Warn("Error while writing response", mlog.Err(err))
    }
}
```

All Coverage Combined

<https://app.codecov.io/gh/vlussenburg/mattermost-server/blob/master/server/channels/api4/channel.go#L378>

github.com/mattermost/mattermost-server/v6/server/channels/api4/channel.go (35.6%)

```
not tracked not covered covered
}
func restoreChannel(c *Context, w http.ResponseWriter, r *http.Request) {
    c.RequireChannelId()
    if c.Err != nil {
        return
    }

    channel, err := c.App.GetChannel(c.AppContext, c.Params.ChannelId)
    if err != nil {
        c.Err = err
        return
    }
    teamId := channel.TeamId

    auditRec := c.MakeAuditRecord("restoreChannel", audit.Fail)
    defer c.LogAuditRec(auditRec)
    auditRec.AddEventPriorState(channel)

    if !c.App.SessionHasPermissionToTeam(*c.AppContext.Session(), teamId, c.SetPermissionError(model.PermissionManageTeam)) {
        return
    }

    channel, err = c.App.RestoreChannel(c.AppContext, channel, c.AppContext)
    if err != nil {
        c.Err = err
        return
    }

    auditRec.AddEventResultState(channel)
    auditRec.AddEventObjectType("channel")
    auditRec.Success()
}
```

github.com/mattermost/mattermost-server/v6/server/channels/api4/channel.go (19.7%)

```
not tracked not covered covered
}
func restoreChannel(c *Context, w http.ResponseWriter, r *http.Request) {
    c.RequireChannelId()
    if c.Err != nil {
        return
    }

    channel, err := c.App.GetChannel(c.AppContext, c.Params.ChannelId)
    if err != nil {
        c.Err = err
        return
    }
    teamId := channel.TeamId

    auditRec := c.MakeAuditRecord("restoreChannel", audit.Fail)
    defer c.LogAuditRec(auditRec)
    auditRec.AddEventPriorState(channel)

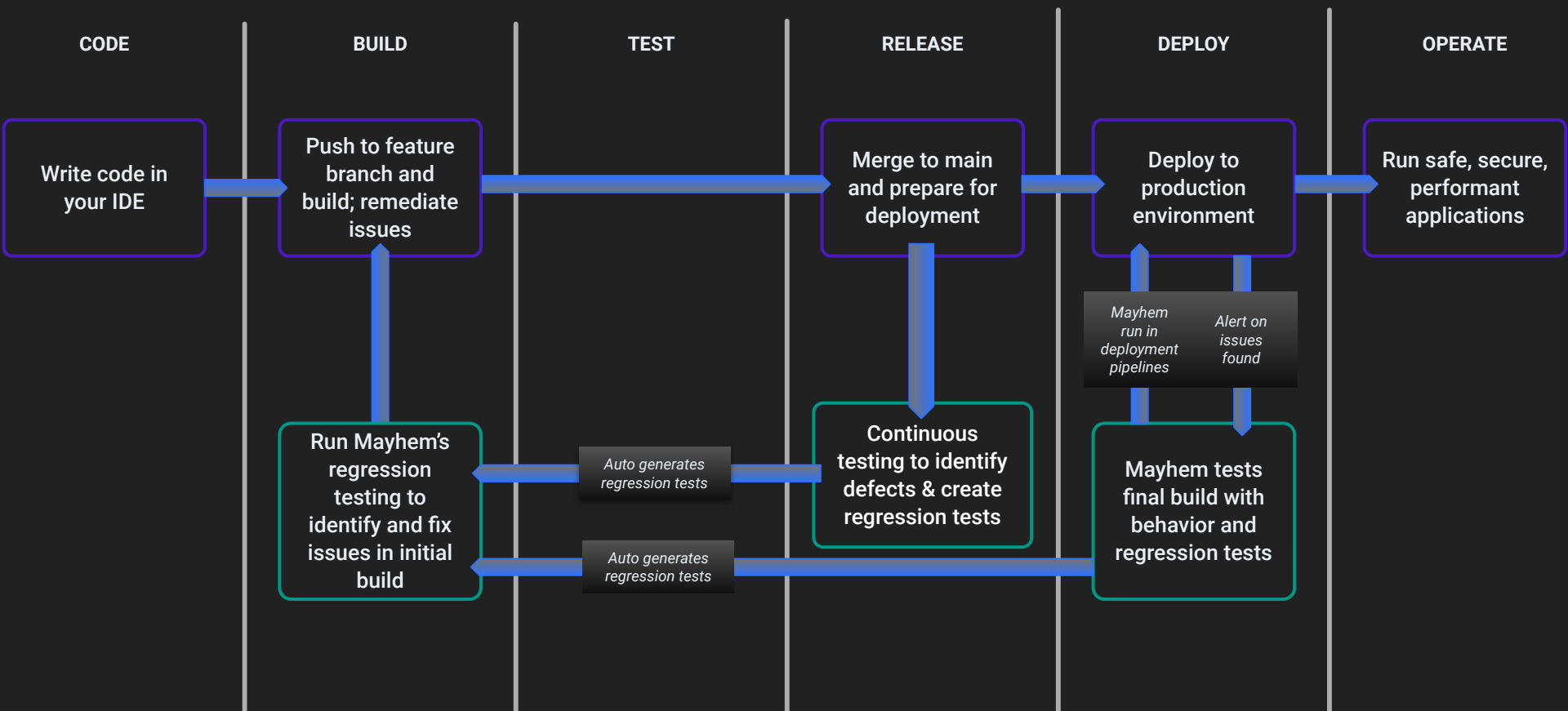
    if !c.App.SessionHasPermissionToTeam(*c.AppContext.Session(), teamId, c.SetPermissionError(model.PermissionManageTeam)) {
        return
    }

    channel, err = c.App.RestoreChannel(c.AppContext, channel, c.AppContext)
    if err != nil {
        c.Err = err
        return
    }

    auditRec.AddEventResultState(channel)
    auditRec.AddEventObjectType("channel")
    auditRec.Success()
}
```



Mayhem in Your Development Pipeline



Conclusion

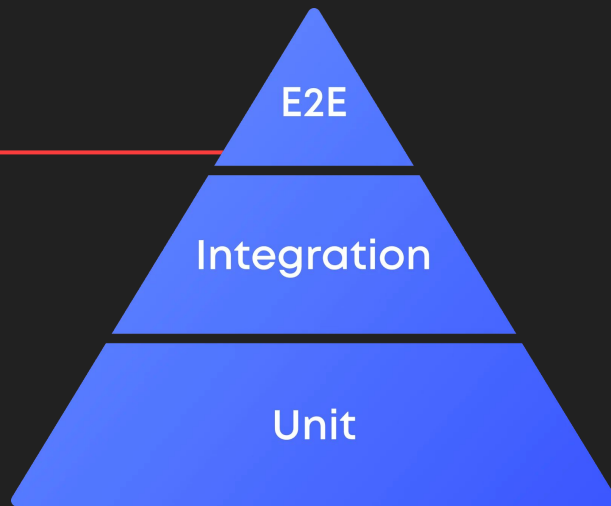
Test coverage helps to see how code is covered by different test types

Testing positively and negatively both is essential

Different approaches make sense for different test types, and you need different test types to do different things

Mayhem allows you to run high coverage low effort “negative tests”

High test coverage derived from multiple test types gives you more confidence in your code



Questions?



Thank you!

Get started for free with Mayhem:

<https://get.mayhem.security/get-mayhem-api>

Contact me:

@vlussenburg on Twitter/Github

@Vincent Lussenburg on forallsecure-mayhem.slack.com

vincentlussenburg on LinkedIn